NASA TECHNICAL NOTE NASA TN D-3924

c. 1

NASA TN D-3924

A REQUEST ORIENTED INFORMATION SELECTION PROGRAM

by Elizabeth Ryan

Lewis Research Center
Cleveland, Ohio

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • APRIL 1967

NASA TN D-3924

A REQUEST ORIENTED INFORMATION SELECTION PROGRAM

By Elizabeth Ryan

Lewis Research Center
Cleveland, Ohio

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

For sale by the Clearinghouse for Federal Scientific and Technical Information
Springfield, Virginia 22151 – CFSTI price $3.00

A REQUEST ORIENTED INFORMATION SELECTION PROGRAM

by Elizabeth Ryan

Lewis Research Center

SUMMARY

The program is a general purpose information retrieval program which can be used with any file of fixed format documents. The program is easily used by noncomputer personnel and provides flexibility in search requests and output format. These features together with efficient design enable the program to satisfy users' requests for information promptly and inexpensively. In order to ensure easy adaptation of the program to change in computer configuration or manufacturer, it is written entirely in FORTRAN IV.

INTRODUCTION

The original need for an information retrieval system at the Lewis Research Center arose in the Office of Reliability and Quality Assurance. The office had obtained magnetic tapes which contained failure and consumption reports and inspectors' reports on parts used by the Atlas and Centaur projects. Through a 5-year period, over 300 000 such reports were collected. The report forms were of a fixed format, each including 44 fields of information describing the part involved and the reason for the report. A detailed description of the reports appears in appendix A. Personnel of the office needed the ability to search through all the report forms, recognize reports on failures of a particular kind, extract some of the information from those reports, and organize the extracted information into a meaningful format.

In planning the program, five objectives were defined:

(1) The input for the program should be of a type easily prepared by someone unfamiliar with computers. In particular, the statement of the search request should not have an artificial appearance.

(2) The program should be flexible both with respect to input and output. The user should have the ability to express very specific search requests so that unmanageable amounts of output are not produced. Furthermore, he should have the ability to specify exactly what information is to be printed from each report and the format in which it is to be printed, as well as to require that the output be sorted into an order which will aid

him in interpreting the results of his search.

(3) The program should be efficient. The total processing time for a request, including tape reading, document inspection, and copying should not be significantly greater than the tape-read time.

(4) Response to a user's need for information must be prompt. In order to ensure this, the program must be compatible with standard operating procedures so that the time span between the user's submittal of a request and the initiation of program execution can be small.

(5) The program should be neither machine nor system dependent. It should be in a form which can easily be modified to adapt to change in computers or to permit expansion of the program's capabilities.

GENERAL DESCRIPTION

Before beginning the discussion of how it is used, a general description of the program may be helpful.

The reports to be searched are stored on magnetic tapes which will be referred to as the search tapes. At the beginning of each search tape is a file of information which describes the format of the reports on the search tape. This will be called the directory file. (Its detailed description can be found in appendix B.) The program begins by reading the directory file in order to prepare itself for the particular kind of documents on the search tape. The directory file completely describes the documents on the search tape. In order to implement the program for a new application, therefore, one must simply prepare search tapes whose directory files describe the new documents to be used, for example, personnel records, library files. The format for search tapes can be found in appendix C. The user's input cards which contain the search and output requests are read and interpreted next. After this, the actual search can begin. Documents are read from the search tape and compared to the user's search request. When a document is found to match the request, it is copied onto an intermediate tape which will be called the hit tape. This process continues until all documents from the search tape have been checked. The documents which were accumulated on the hit tape are then read, and the requested information fields are copied from them. The copied information is sorted and formatted, according to the user's output request, and printed.

The hit tape is written in a format very much like the search tape. A directory file and a copy of the search request which produced the hit tape are placed at its beginning. The hit tape can, therefore, be saved for use later on as a search tape. This effectively gives the user the ability to produce a relatively general subfile of documents on some subject and to perform more specific searches on the subfile as particular aspects of its subject become of interest. A description of the hit tape's format can be found in appendix C.

INPUT TO PROGRAM

The major input to be designed was the description of the reports to be selected, the statement of the search criteria. The user needed the ability, for example, to look for reports on parts numbered exactly MX472-35 or for reports with a report number greater than or less than some number. The ability to find all reports related to VALVES, whether the part name was INTAKE VALVE, VALVE ASSEMBLY, or simply VALVE was required. Also necessary was the ability to combine requirements; for instance, to find reports on only those TRANSISTORS which had been manufactured by a particular company, or to find all reports on TRANSISTORS manufactured by company A and simultaneously to find all reports on VACUUM TUBES manufactured by company B. He also needed to be able to list several acceptable entries for a field, to find reports on parts manufactured by any of several companies. An input form which would include all these capabilities and still be concise and explicit was needed. The following is a description of the form which was devised:

(1) A term of the search statement will be defined as the name of a field connected to a value by any of the relational operators: greater than, less than, equal to, not equal to, greater than or equal to, less than or equal to. By "value" is meant the sought after entry in some field (i.e., TRANSISTOR, 5/12/64, MX472-53, 30014, are possible values for the fields giving part name, report date, part number, and report number, respectively). The relational operators are written as follows:

greater than	GT
less than	LT
equal to	=
not equal to	NE
greater than or equal to	GE
less than or equal to	LE

Examples of terms are the following:

F/I PARTNO = 37ABX29 The value entered in the field F/I PARTNO must be exactly 37ABX29.

FAIL DATE \$GT\$ 1/1/62	The value entered in the field FAIL DATE must be greater than 1/1/62 (i.e., must be a date later than 1/1/62).

(2) The search statement is formed by connecting several terms with the following logical operators:

both terms must be true	\$AND\$
either or both terms must be true	\$OR\$
the term must not be true	\$NOT\$

Examples are as follows:

F/I NAME = TRANSISTOR \$AND\$ F/I MFR = A	The value in field F/I NAME must be TRANSISTOR, and also the value in field F/I MFR must be A. This statement describes transistors manufactured by company A.
F/I NAME = TRANSISTOR \$OR\$ F/I NAME = XSTR	Either the value in field F/I NAME is TRANSISTOR, or the value in field F/I NAME is XSTR. This statement describes parts named either TRANSISTOR or XSTR.

(3) A series is defined as a field name connected by a relational operator to two or more values each of which is connected by the logical operator \$OR\$. Its meaning is that any of the values in the series is an acceptable entry in the specified field. For example,

F/I MFR = A \$OR\$ B \$OR\$ C

This statement describes parts manufactured either by companies A, B, or C.

(4) In combining terms the normal hierarchy for logical operations is observed, (i.e., operations are performed in the order NOT, AND, OR). Order may be made explicit by using parenthesis. For example, assume that the user wishes to find all reports about transistors manufactured by company A or company B. His request could correctly be stated as follows:

(F/I MFR = A \$OR\$ F/I MFR = B) \$AND\$

F/I NAME = TRANSISTOR

The parentheses specify that the \$OR\$ is to be performed first and its result bound by the AND. The statement without the parentheses,

F/I MFR = A OR F/I MFR = B AND F/I NAME = TRANSISTOR

would not express the user's intention, since it would describe transistors manufactured by company B, but also parts of any kind manufactured by company A. (Since the AND is performed first, it will bind the F/I NAME condition only to the term F/I MFR = B, and that result will be connected by the OR to F/I MFR = A.)

(5) The series is treated as a single term when being combined with other terms. The example in item (4) can therefore also be correctly stated as

F/I MFR = A OR B AND F/I NAME = TRANSISTOR

Since the previous form is considered to consist only of two terms, there can be no question as to what the AND is binding.

(6) The first or last n alphameric characters in a field can be checked by inserting an F(first) or L(last) immediately after the leading $ of the relational operator. If relation is "equal to," the graphic = must be preceded by a $F or $L. The n characters which are being searched for are the value in the term. For example,

F/I NAME $L = VALVE

This described all reports having the characters V-A-L-V-E as the last five nonblank characters in the F/I NAME field.

In addition to stating the search criteria, the user must be able to describe the output for each run. He needs the ability to specify not only which fields of information are to be printed, but in what order across the page they should be printed. In order to obtain meaningful printouts, he must also have the ability to request that the selected reports appear in some order. He might wish, for instance, to request that the reports appear in order according to the values in their part name field, and furthermore, that within that order they be ordered according to the values in the manufacturer field. Since the reports are to be ordered by these two fields, the user will probably want them to appear leftmost on the printout. He expresses these specifications simply by listing the report fields he wishes to see in the order in which they should appear across the page and by listing the fields by which the output is to be ordered.

The options outlined previously are expressed by the user through these four control cards:

(1) *SEARCH This card contains the statement of the search critieria written according to the rules defined previously. For example,

*SEARCH F/I NAME = TRANSISTOR OR XSTR AND

FAIL DATE GT 1/1/61 OR F/I NAME =

DIODE AND ACTIVITY = A9

The user is interested in all "transistors" or "xstrs" which were reported since January 1, 1961. He also wishes to see all reports about "diodes" which were being used in activity "A9."

(2) *LIST This card lists the names of all report fields to be printed as output. The report fields will appear as tabular columns in the order left-to-right in which they appear on the LIST card. This card may be omitted, in which case only a statement of the number of reports found, will be printed as a result of the search. For example (Note that the names are separated by commas.),

*LIST F/I NAME, F/I PART NO, FAIL DATE, ACTIVITY,

MISSILE SERIES, MISSILE TYPE CODE,

F/I MFR CODE, N/A NAME

The output produced by this request can be seen in appendix D.

(3) *SORT This card contains the names of the report fields by which the tabular output will be sorted. Alphabetic fields will appear in normal alphabetic order, while numeric fields appear in ascending numeric order. The first field listed will be used as the primary key; if a second field is listed, it will be used as the secondary key. It should be separated from the first field by a comma. Only two keys can be specified. The SORT card may be omitted, in which case the reports will appear in the order they were encountered on the search tape. For example,

*SORT F/I NAME, F/I PARTNO

Note the ordering of the sample output (appendix D) which was specified by this card.

(4) *END — This card simply marks the end of input and must appear as the last card of every input deck.

The following rules apply to the format of all the control cards:

(1) An asterisk must appear in column 1 of each control card. It is followed by the card identifier (SEARCH, LIST, SORT, or END). No blanks may appear within the identifier, but it may be separated from the asterisk by any number of blanks. (The END card in the sample input of appendix D illustrates this.)

(2) The specification information for each card (the search statement for the SEARCH card or the field names for the LIST and SORT cards) is separated from the identifier by at least one blank column. Blanks may appear anywhere within the specification information.

(3) The contents of the control cards may be continued onto additional cards at any point. (This is illustrated by both the SEARCH and LIST cards in the sample input deck, appendix D.) The continuation cards must not have an asterisk in column 1.

(4) The control cards may appear in any order except that the END card must be last.

PROGRAM OUTPUT

Printed output consists of the following sections:

(1) Input analysis - Before the search can begin, the user's input must be read and analyzed. The first output to appear on the listing is associated with this input analysis.

If the search tape for this run is a hit tape which was saved from a previous search, the search statement which produced the hit tape is printed preceded by the following statement:

SEARCH TAPE CONSTRUCTED OF ITEMS AS FOLLOWS

As each control card is read and interpreted, any errors found in its format will be indicated by appropriate messages. (See appendix E for a listing and explanation of all error messages.)

When the SEARCH card is encountered, it is printed out exactly as it was read. Immediately after this an analysis of the search request is printed. This is produced by breaking the search statement into simpler substatements. Each substatement is assigned a numeric label. The way in which the substatements are logically related to form the search statement is then indicated. (See sample output of appendix D for an example of a search statement and its analysis.) If any logical redundancies or inconsistencies are detected during analysis of the search statement, these will be indicated by error messages.

(2) Search results - When the search has been completed, its results are reported to the user. The number of reports which were found is reported in the following form:

SEARCH YIELDS XXXXXXX HITS.

This statement always appears following completion of the search.

If a LIST card was included, column headings will be produced and all report fields requested will be printed in tabular form. Output will be ordered according to the user's request on the sort card.

If the total width of the report fields requested on the LIST card exceeds the page width, right-most fields will be omitted. For the program application being described, however, an exception was made for the description fields (DESC LINE 1, DESC LINE 2, DESC LINE 3). Provided that the user lists these fields as the last fields on the LIST card, they are printed as two additional lines per report.

If output sorting is requested, it may be necessary to split the reports into two or more sorted groups for output. (The conditions which force this to occur are described in the section on subroutine SOUT, appendix F.) If so, this will be indicated by messages of the form:

SELECTED REPORTS WILL BE PRINTED IN XXXXX GROUPS.

. . . GROUP XXXX YYYYY REPORTS

(See sample output, appendix D).

EFFICIENCY

Execution times for all searches recorded thus far have proved to be a function only of the length of the search tapes. The complexity of the search statement has not been a factor in determining run times.

For the current application, approximately 33 000 report forms are stored on a single tape. At 800-bit-per-inch density, the tapes are one-half to three-fourths full. For the computer configuration in use at this installation, the program searches such a tape in approximately 3 minutes.

The efficiency of the program is due in large part to two factors, the analysis of the search statement and input-output optimization.

The search statement is analyzed thoroughly as it is read from the SEARCH card. The analysis has three purposes.

First, it determines the most efficient possible order of operations to be used in comparing each report against this particular search statement.

Secondly, it reduces the search statement to a set of logical switches which will be used to channel the search algorithm through the ordered operations just determined. When the reduction is complete, the search statement as expressed by the user exists only in the form of these switches.

Third, as the statement is reduced, logic errors and redundancies may be detected. Redundant requirements can be omitted, thus eliminating unnecessary checking at search time. Logic errors may be such as to render the entire statement meaningless, in which case the search will not be carried out.

In order to optimize input-output speed, the subroutines required for the three distinct program phases (input analysis, search, and reporting) are overlaid. By thus minimizing the core storage devoted to program, a maximal amount of core can be devoted to input-output buffers. This, in turn, allows a greater amount of information to be transmitted for each input-output operation.

Prompt Response

Because the program is written entirely in FORTRAN IV, it can be run as a normal job at most installations. As soon as the user's request has been prepared, therefore, it can be inserted in the job stream for processing. This fact, together with the efficiency of the program itself, ensures that response to a search request will be prompt.

Expansion and Modification

Since FORTRAN IV is a language widely used among computer installations and widely supported by computer manufacturers, the program can be easily modified by the user installation. The program is separated logically into subroutines so that one program function can be changed or rewritten without affecting another.

INTERNAL ALGORITHMS

This section will describe the algorithms developed for analysis of a search statement and for output ordering.

Analysis of Search Statement

The search statement is essentially a logical expression written using the conventions of an algebraic notation. Each term of the statement corresponds to a logical variable, a series to a number of such variables bound together with the logical operator OR. Thus, in a symbolic logic notation, the statement

F/I NAME = TRANSISTOR OR XSTR AND FAIL DATE GT 1/1/61

OR F/I NAME = DIODE AND ACTIVITY = A9

could be represented by the following expression:

$$(q_1 + q_2)q_3 + q_4q_5$$

where

q_1	represents the "truth" statement	F/I NAME is TRANSISTOR
q_2	represents the "truth" statement	F/I NAME is XSTR
q_3	represents the "truth" statement	FAIL DATE is later than 1/1/61
q_4	represents the "truth" statement	F/I NAME is DIODE
q_5	represents the "truth" statement	ACTIVITY is A9

The first part of the analysis of the statement is concerned with the algebraic reduction of its logical expression. This reduction is analogous to the algebraic reduction of a mathematical expression and can be handled in much the same manner.

The technique used for this phase of analysis is one developed by Peter B. Sheridan for use in the IBM 704 FORTRAN II arithmetic translator. It is described in reference 1. Sheridan is concerned with the reduction of an expression involving operators of four precedence levels (function, exponentiation, multiplication-division, addition-subtraction). The search statements involve operators of three levels (NOT, AND, and OR). The AND operator can conveniently be considered the analog of multiplication and the OR of addition. The NOT operation has the highest level of precedence, and could therefore, be considered the analog of the function operator. It is more similar in context, however, to the minus sign used as an unary operator to mean algebraic negation (e.g., X = SIN($-\theta$)). This sense of the minus sign is handled by Sheridan in a somewhat artificial manner since he must also provide for the use of minus as the binary operator ''subtract'' (e.g., $Z = X - Y$). (Note that the unary operator negate has a level of precedence equal to that of function, while the binary operation subtract has a level of precedence equal to that of addition.) Because the NOT operator matches the unary minus not only in its level of precedence but in its context within a statement, it was decided to handle NOT as Sheridan handles the minus sign rather than as a function.

Use of Sheridan's algorithm is discontinued after the telescoping and ordering of segments. At this point, there has been formed an array P which is composed of triples of the form (C, Z, Y). A triple has been formed for each term of the search statement; some additional triples represent combinations of those terms. The first element C of each triple is an indicator of the order in which the triple is to be considered relative to other triples. The operands Y of all triples having equal C elements are to be combined using operator Z which may represent NOT, AND, or OR. The array P is constructed as follows: For the ith triple, $P_{1i} = C_i$.

Triples of high precedence have low C_i; those of low precedence have high C_i. Within P, the triples are ordered by ascending C_i. All triples with equal $C_i = K$ are said to belong to segment K.

$P_{2i} = Z_i$ where Z_i is an integer indicating the operator associated with triple i. Since no two of the possible operators (NOT, AND, and OR) have the same level of precedence, all triples within a given segment will have the same operator.

$P_{3i} = Y_i$ where Y_i, the operand, may refer to another segment or to a variable - one of the original terms. A segment whose operator is NOT must be of length 1 since NOT is a unary operator. Segments whose operators are either AND or OR may be of any length because of the associativity of the operations. Because AND and OR also exhibit commutativity, the order in which the triples of such segments are combined is immaterial.

To illustrate, the following is a sample search statement:

F/I NAME = TRANSISTOR OR XSTR AND FAIL DATE GT 1/1/61

OR F/I NAME = DIODE AND ACTIVITY = A9

or

$$(q_1 + q_2)q_3 + q_4q_5$$

is represented at this point in analysis by the following table:

	Triple							
	1	2	3	4	5	6	7	8
C	1	1	2	2	5	5	14	14
Z	OR	OR	AND	AND	OR	OR	AND	AND
Y	2	14	$3	5	$2	$1	$5	$4

The operators NOT, OR, and AND are represented by Z = 3, 2, and 1, respectively. The numeric operands (Y = 2 for triple 1) refer directly to the corresponding segment. The flagged operands ($2 for triple 5) are pointers to three additional tables which define the original "truth statements" q_1 - q_5. These three arrays appear as follows:

Statement	1	2	3	4	5
NOUN	12	12	23	12	2
PRED	=	=	>	=	=
OBJ	1	4	7	8	11

The contents of PRED shown graphically here are actually numeric indicators (i.e., = would be represented by 1; > by 3). The numbers in NOUN relate to a list of all the report fields; field 2 is ACTIVITY, field 12 is F/I NAME, and field 23 is FAIL DATE. OBJ contains pointers to array TABL which contains the values associated with each term as shown in the following table:

TABL	1	000000
	2	00TRAN
	3	SISTOR
	4	000000
	5	000000
	6	00XSTR
	7	010101
	8	000000
	9	000000
	10	0DIODE
	11	0000A9

The expanded contents of P are printed out at this point in reverse order. As each segment is expanded for output, it is checked for logical redundancies and errors; if either or both are found, a comment will be printed below the segment in question. (For additional comments, see appendix E.) When expanded, the aforementioned array P will be printed as follows:

ANALYSIS OF SEARCH STATEMENT

(014 F/I NAME = 0000000000000DIODE AND ACTIVITY = 0000A9

(005 F/I NAME = 00000000TRANSISTOR OR F/I NAME = 00000000000000XSTR

(002 (005 TRUE AND FAIL DATE GT 010101

(001 (014 TRUE OR (002 TRUE

ACCEPT IF (001 IS TRUE

This can be seen to be a correct restatement of the original search statement. If each report form on the search tape were checked in the manner indicated (i.e., using the entire P array in reverse order), correct results would be obtained. The remainder of the analysis is concerned with developing from this a "most efficient" plan for report checking. The aim is to develop an array of "pointers" which will channel the report checking function through the requirements of the search statement in an efficient order.

There are two important differences between the evaluation of an arithmetic expression and the evaluation of a logical expression.

The first is that, in the case of an arithmetic expression, each term must be considered in order to arrive at a correct value for the expression. In contrast, the value of a logical expression, which can be only TRUE or FALSE, can often be determined without considering all its terms. For example, consider the following algebraic expression:

$$v = px(q + s)$$

If p, q, and s represent numeric values, say 4, 9, and 3, respectively, the value of each must be known before arriving at the correct numeric value, 48, for v. On the other hand, if p, q, and s represent logical values and the operators x and + represent the logical operators AND and OR, respectively, it may not be necessary to consider the value of all three variables. Suppose p has the value FALSE, then it can be seen immediately that v also must have the value FALSE. The values of q and s need not be considered.

The second difference between arithmetic and logical expressions is concerned with the order of operations performed in their evaluation. If the expression is numeric, it is most efficient to completely evaluate the inner expressions first and then combine their values with outermost terms. If the expression is logical, however, it is most efficient to examine the outermost terms first. In this way, the inner subexpressions may never have to be evaluated.

These two characteristics can be used to advantage in determining an affective sequence of operations for report checking. The search controlling array STEP must first cause outermost terms to be considered before inner ones; and secondly, it must ensure that only necessary terms will be considered. In order to fulfill the first requirement, the pass through array P, which prints the search analysis and checks the logic of each segment, performs two additional functions:

(1) The cumulative length of each segment is recorded in an array, LIST. By cumulative length of a segment is meant the sum of the number of its triples whose operands are flagged and the cumulative lengths of all segments which appear as operands of triples in the segment.

(2) The triples within each segment are reordered. Those triples whose operands are flagged are placed at the beginning of each segment. The remaining triples, whose operands refer to other segments, are placed in ascending order according to the cumulative length of the operand segment. The effect of this is to place triples with simple operands ahead of those whose operands are more complex.

Note that both these functions can be performed on the single back to front pass through P. This is true because any segment referred to by another must be an "inner" expression and consequently have a higher C number. Its length will, therefore, have been determined before the segment which refers to it is encountered.

After this reordering has taken place, the P array for the example being used will be

	Triple							
	1	2	3	4	5	6	7	8
C	1	1	2	2	5	5	14	14
Z	OR	OR	AND	AND	OR	OR	AND	AND
Y	14	2	$3	5	$2	$1	$5	$4

Essentially, the logical expression

$$(q_1 + q_2)q_3 + q_4q_5$$

has been rewritten

$$q_4q_5 + q_3(q_1 + q_2)$$

Since P has now been arranged to ensure that triples will be considered in an efficient order, the remaining requirement is to ensure that no triple will be considered unnecessarily. To accomplish this, STEP is constructed as a two column array. Corresponding to each triple in P, there will be defined an entry in column 1 of STEP which is to be consulted by the report checking routine if the triple's operand is found to be TRUE. The entry will be either equal to 0, which indicates that the report has now been shown not to match the search statement, equal to -1, which indicates that the report has now been shown to match the search statement, or equal to some positive number, which points to the next triple within P which should be checked. Column 2 of STEP is similarly defined and will be consulted if the triple's operand is found to be FALSE.

The construction of STEP can best be defined by referring again to the example. Beginning at the top of P the operand of the first triple is the value of segment 14. In checking a report from the search tape, then, the first question would be whether or not the conditions of segment 14 are satisfied. Since the first triple of segment 14 appears as triple 7 in P, a variable START is equal to 7.

Segment 14 is an AND segment; therefore, if any triple is determined to be false, the entire segment is false - no other triples need to be checked. On the other hand, if a triple is found to be true, the next triple must be checked until all have been found to be true. Therefore, the subscript of the next triple is placed in the TRUE column of STEP opposite triple 7. If the segment is false, the significance of this is determined by the type of segment 1 which referred to segment 14. Since segment 1 is an OR segment, the falsity of one of its triples means only that the next triple must be checked. Therefore, the entry in the FALSE column of STEP for triple 7 points to the first triple of segment 2. (The logical paths which trace the relatedness of the various segments are recorded during the construction of STEP, in a push-down list PATH.)

By similar reasoning, since triple 8 is the last triple of the AND segment 14, it will cause the entire segment 14 to be true if it is true. When this is related back to segment 1, which is an OR segment, it is seen that segment 1 must also be true. Since segment 1 is the outermost segment - it is referred to by no other segment - its truth is a necessary and sufficient condition for the acceptance of the report being checked. Thus, the correct entry for the TRUE column of STEP for triple 8 is a -1.

The remainder of the STEP table is filled in, according to the same criteria. At the conclusion the variable START = 7 and the STEP table is as follows:

Triple	Step	
	True	False
1	--	--
2	--	--
3	5	0
4	--	--
5	-1	6
6	-1	0
7	8	3
8	-1	3

Referring back to the five original truth statements of the search statement, it can be seen that STEP constitutes a program-generated flow chart (see fig. 1) of the strategy for comparing a document to the search statement of the example.

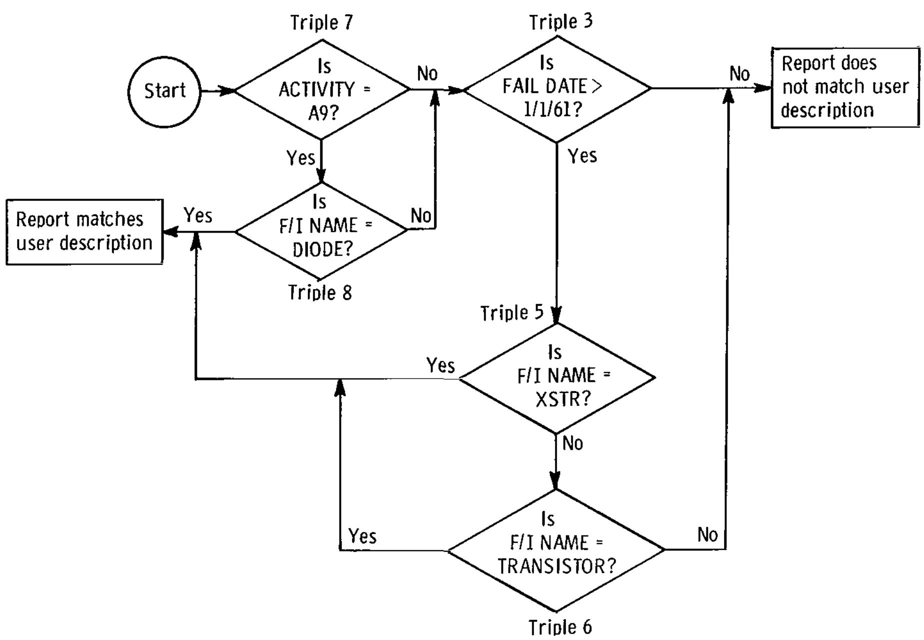

Figure 1. - Program developed search logic

Since the report checking subroutines simply follow along the control paths defined by the analytical routines, the search operation becomes very efficient. No decision making is required except for the simple comparison of report field to user-requested value. The number of such comparisons is minimized by the optimization of the "flow chart" defined for a particular search statement.

Output Ordering

The tabular output is ordered by the two subroutines, SOUT and SORTER. In subroutine SOUT the process is initiated by packing the required fields (those to be used as sort keys and those to appear as output) into a buffer area. For each document, the fields are placed in the buffer as follows.

The fields to be used as sort keys are extracted and stored first. The fields are right-adjusted within the buffer words, but the adjustment of the values within the fields is not disturbed. Blanks which appear in the values are replaced by zeros. Thus, the sorting algorithm assumes that numeric values are right-adjusted and alphabetic values are left-adjusted in their fields as documents are represented on the search tapes. The length, in words, of the sort fields (there may be one or two) is given by the variable KLEAR.

The fields requested as output are extracted and placed in the buffer following the sort fields. The output fields are left-adjusted in the buffer words. The total number of buffer words required per document for both sort and output fields is given by variable LENGTH.

As the sort fields from each document are extracted and stored, they are compared to the sort fields of the previous document. Consecutive documents whose sort fields are found to be in ascending order are said to comprise a naturally ordered group of documents. (A group may include only one member.) A count of the number of such naturally ordered groups is maintained in variable K. The location and length of each group is recorded in the two arrays, LIST1 and LIST2. For group J, the Jth entry in LIST1 gives the beginning buffer word of the first document in the group, and the Jth entry in LIST2 gives the beginning buffer word of the first document in the next group. (Initially then, if $J \neq 1$, LIST1, = LIST2_{J-1}.)

When the buffer area BUFFER has been filled, subroutine SORTER is called to complete the ordering and printing of output.

The sort keys from the first member of each of the K groups are compared by SORTER to find the smallest key. During the K comparisons, each time a new smallest key is found the number of the group which produced it is recorded in a push-down list ISAVE. When all K groups have been checked, the output fields for the member which produced the smallest key can be printed.

Suppose the Ith group contained the smallest key. After the first member of the group has been printed, LIST1_I is incremented by LENGTH so that it gives the location of the next member. The previous smallest key can be found using the information stored in ISAVE. It needs only be compared to the key of the new first member of group I and the first members of groups I to K. Again, the member whose key is smallest is printed, LIST1 is incremented, and a new series of comparisons is initiated depending upon the status of ISAVE.

Each time a LIST1 entry is incremented to point to a new member of some group I, the resulting $LIST1_I$ is compared to $LIST2_I$. If they are equal, group I has been printed completely and contains no additional members. In this case, $LIST1_I$ is set to zero so that SORTER can easily recognize, in succeeding scans, that group I should be skipped. If I = K (i.e., if the group which just became empty is the last group), K is reduced by one. Similarly, a variable IBEG is used to indicate the first group; if group IBEG becomes empty, IBEG is incremented by one. If all groups except one become empty during the ordering process, the members of the remaining group are simply printed one after the other as they lie in BUFFER.

When all groups become empty, the output of one ordered batch is complete and control returns to SOUT. If additional batches are required, SOUT will reinitiate the process for the new set of documents (See section SOUT in appendix F.).

Lewis Research Center,
National Aeronautics and Space Administration,
Cleveland, Ohio, January 25, 1967,
125-23-02-10-22.

APPENDIX A

DESCRIPTION OF REPORT FORMS

The failure and consumption data and inspectors reports which make up the search file for the application being described contain the following information fields:

(1) REPORT NO	Seven character serial number which uniquely identifies each report
(2) ACTIVITY	Two character code which identifies the base, installation, or subcontractor at which the report originated
(3) MISSILE TYPE CODE	Four character code which is assigned by the government to a contractor or by a contractor to the contract under which the part was used
(4) MISSILE SERIES	One character which identifies the missile program involved
(5) MISSILE SERIAL NO	Three character serial number which identifies the specific missile involved
(6) TGSE-GSE PART NO	Twelve character part number of the test ground support equipment or ground support equipment which contained the part involved
(7) TGSE-GSE SERIAL NO	Seven character serial number of the test ground support equipment or ground support equipment which contained the part involved
(8) F/I PART NO	Nineteen character part number of the part involved
(9) F/I SERIAL NO	Seven character serial number of the part involved
(10) F/I NAME	Thirteen character name of the part involved
(11) F/I REFERENCE DES	Seven character reference indicating the position of the part on a schematic
(12) N/A PART NO	Nineteen character part number of the next higher assembly in which the part was used
(13) N/A SERIAL NO	Seven character serial number of the next higher assembly
(14) N/A NAME	Twelve character name of the next higher assembly
(15) FAIL CODES	Six character code which gives the cause and effect of a failure

(16 F/I MFR CODE — Five character government-assigned code which identifies the manufacturer of the part involved

(17) SUBSTITUTE PART NO — Nineteen character part number of the alternate part used as a replacement

(18) REPL SERIAL NO — Seven character serial number of the replacement part

(19) SYSTEM — Two character number of the airborne or ground support system in which the part was used

(20) FAIL DATE — Date of failure or replacement is recorded as five characters of the form $YM_1M_2D_1D_2$ where Y is the last digit of the year, M_1M_2 is the month, and D_1D_2 is the day of the month

(21) USAGE 1-2 — Sixteen characters which indicate the total operating time to replacement of the part involved; "time" may be expressed in (1) hours-minutes-seconds, (2) cycles, (3) days, or (4) miles

(22) DDCODE — One character code which indicates whether a failure was discovered during test, inspection, shipping, operation, and so forth

(23) RRCODE — One character code which indicates the reason for the report (i.e., failure, engineering change, etc.)

(24) RACODE — One character code which indicates the repair action taken on a failed part

(25) RBCODE — One character code which indicates the replacement action taken

(26) FAIL CLASS — One character code which indicates the relative importance of a failure

(27) RESP CODE — Two character code which indicates the primary cause of a failure

(28) FAIL CATEGORY — One character which indicates the general point at which a failure occurred (i.e., testing, countdown, etc.)

(29) STD/CPLX — Two character code which indicates the offsite stand or complex in use when the failure or replacement occurred

(30) HV GRP NO — Two characters which are used to group critical parts into general part or material categories

(31) REV IND.	One character which identifies a form as a revision of a previous report
(32) IR	One character which indicates that a form is an inspector's report
(33) WEEK CODE	Three character code which identifies the current week or processing period
(34) DESC LINE1	Forty-five characters which make up the first line of commentary on any unusual circumstances associated with the report
(35) DESC LINE2	Forty-five characters which make up the second line of commentary on any unusual circumstances associated with the report
(36) DESC LINE3	Forty-five characters which make up the third line of commentary on any unusual circumstances associated with the report
(37) REPORTING DEPT	Four characters which identify the department which submitted the report
(38) RECEIVING PRT NO	Seven character number of a receiving report
(39) ASGD DEPT	Four characters which identify the department assigned to investigate a failure
(40) QUANTITY REJECT	Five characters which specify the quantity of parts rejected (if this report originated in inspection or test)
(41) SOURCE	One character which indicates the source of the part involved
(42) MULT USE IND	One character
(43) F/I SERIAL CODE	One character which signifies that more than one of the indicated parts was involved in a failure
(44) FINAL DISPOSITION	One character which indicates the final disposition of the part involved

In addition to the aforementioned fields which appear on the original report documents, the contractor who prepared the search tapes added to the tapes four code fields which facilitate the use of the tapes under the 9 PAK system. The additional fields are not relevant to this report.

APPENDIX B

FORMAT OF DIRECTORY FILE

The tapes, as acquired by the Office of Reliability and Quality Assurance, had been prepared for use under the 9 PAK system. The directory file required by 9 PAK contains four types of records which are identified by the appearance of a B, E, F, or P as the first character of each record. (All characters are represented in the IBM internal BCD 6-bit character set. The tapes are written in the binary mode.) Only the F-type records are used by the program; their format is the following:

character 1	"F"
characters 11 to 13	Length, in bits, of a report field
characters 18 to 21	Number of complete 36-bit words which precede this field in the report forms
characters 22 and 23	Number of bits which, in addition to the words given in characters 24 to 27, precede this field in the report forms
characters 24 to 47	Name of the report fields
characters 48 to 78	Ignored by the program

An F-type record must appear for each of the fields which appear on the search tapes. (These are enumerated in appendix A.) They are used to define the length and placement of each field within the report forms. For example, the F record for the ACTIVITY field is the following:

Characters						
1 to 6	F	0	1	0	0	0
7 to 12	4	1	1	0	0	1
13 to 18	2					0
19 to 24	0	0	2	0	6	A
25 to 30	C	T	I	V	I	T
31 to 36	Y					
37 to 42						
43 to 48						0
49 to 54	3	1				
55 to 60			9	9	9	9
61 to 66	9	9				
67 to 72	Y	L	(	C		2
73 to 78	.					

Word	
1	0 1 0 0 0 0
2	B 0 8 1 4 9
3	2 J A S M 6

It defines the ACTIVITY field as being 12 bits (2 characters) long and as appearing in the position following word 2 bit 6 within the report form. Applying this information to the partial report form, shown at the left as it would appear on tape, reveals that the ACTIVITY in which the reported part was involved is identified as JA.

The secondary files of the search tapes, which contain the report forms, were of the format described by the directory file. Several report forms (usually 15) were packed in a single tape record thus producing tape records of 930 words.

The following modifications were made to the original tapes for use with the program:

(1) The tapes were made FORTRAN readable. This involved two modifications, namely,

(a) Tape records were written as logical records made up of physical records of 255 words or less.

(b) The correct FORTRAN control word was placed at the beginning of each physical record.

(2) An integer giving the number of data words following in the logical record was added as the first data word of each logical record.

(3) FORTRAN readable records were placed on the tape to indicate a following end of file, end of tape condition, or end of set condition. (A search file - the set - may consist of as many as four physical reels. The end of each tape except the last will be marked by the end of tape indicator, the last by an end of set.)

The records have as their first word, the integer 1, indicating that only one data word follows. The data word for each of the three conditions are as follows (shown in octal):

END OF FILE APPEARS NEXT ON TAPE	152631432560
END OF TAPE	156321472560
END OF SET	156225636060

APPENDIX C

FORMAT OF HIT TAPES

Although the 9 PAK type format (described in appendix B) includes the information necessary for processing the search tapes, a different organization of the information is more suitable for the purposes of the program. The hit tapes are, therefore, written in a slightly different format than the original search tapes, and it is recommended that new program applications use the hit tape format exclusively.

The first, or directory file, has the following format: (All records are FORTRAN-readable - the FORTRAN control words, though not shown, are assumed to be present. Records are shown as FORTRAN logical records and may consist of several physical tape records.)

RECORD 1:

word 1	N, the integer number of 36-bit words which follow in this record
word 2	Integer number of 36-bit words which make up one search document (report form)
word 3	Integer number of search documents per (logical) record
word 4	M, the integer number of fields per search document, and, therefore, the number of directory records to follow in this file
words 5 to N + 1	A copy of the search statement (in A1 format) which produced this hit tape (a maximum of five cards will be copied)

RECORDS 2 to M + 1 describe each field of the search documents as follows:

word 1	K, the integer number of 36-bit words which follow in this record
word 2	Length of this field (in bits)
word 3	Number of bits which precede this field in the search documents
word 4	Length (in bits) of the field name
word 5	Beginning word position of this field name in an array which contains all field names
word 6	Ending word position of this field name in an array which contains all field names
words 7 to K + 1	Name of the field (in A6 format); embedded blanks are included

RECORD M + 2 is the end of file marker as defined in appendix B. It is followed by an end of file which separates the directory file from the search documents. The format of the records which contain the search documents is as follows:

word 1	J, the integer number of 36-bit words which follow in this record
words 2 to J + 1	One or more search documents (the number of documents per record will not exceed the number specified in word 3 of directory record 1)

The end of the hit tape is marked by the end of set message defined in appendix B. No provision is made currently for producing multiple-reel hit tapes.

APPENDIX D

SAMPLE JOB

This appendix contains the input for and output from a search which was run using the program. As indicated by the output listing, 509 hits were found during the search. The listing of the hits as requested on the *LIST card is terminated after 54 documents in order to conserve space.

Sample Input Deck

```
*SEARCH  F/INAME = TRANSISTOR  $OR$  XSTR   $AND$   FAIL DATE  $GT$  1/1/61
  $OR$  F/INAME=  DIODE  $AND$  ACTIVITY = A9
*LIST   F/INAME, F/IPARTNO, FAIL DATE, ACTIVITY, MISSILE SERIES,
     MISSILE TYPE CODE, F/I MFR CODE, N/ANAME
*SORT   F/INAME, F/I PARTNO
*         END
```

Sample Output Listing

The following partial listing shows the format of output obtained:

```
*SEARCH  F/INAME = TRANSISTOR  $OR$  XSTR   $AND$   FAIL DATE  $GT$  1/1/61
  $OR$  F/INAME=  DIODE  $AND$  ACTIVITY = A9

ANALYSIS OF SEARCH REQUEST

(014       ACTIVITY       =  0000A9  AND F/I NAME      =  000000000000DIODE

(005       F/I NAME       =  0000000000000XSTR  OR  F/I NAME      =  00000000TRANSISTOR

(002       FAIL DATE     GT  010101  AND  (005  TRUE

(001        (014  TRUE  OR   (002  TRUE

 ACCEPT IF  (001  IS TRUE
```

SEARCH YIELDS 509 HITS.

SELECTED REPORTS WILL BE PRINTED IN 1 GROUPS.

...GROUP 1 509 REPORTS

F/I NAME	F/I PART NO	FAIL DATE	ACTIVITY	MISSILE SERIES	MISSILE TYPE CODE	F/I MFR CODE	N/A NAME
DIODE	1 5M82Z	20822	A9	D	SM65	81223	BUFFER STOR
DIODE	1N1084	20611	A9	E	SM65		SIG MON
DIODE	1/4M30Z5	20920	A9	E	SM65	81223	PWR CTL PNL
DIODE	650C5	21026	A9	D	SM65	96214	SUB ASSY
TRANSISTOR	16T36MP	20723	9D	D	SM65	24446	RECORDER
TRANSISTOR	16T36MP	20725	9D	D	SM65	24446	RECORDER
TRANSISTOR	26-12199- 3	20322	99	D	SM65	96214	
TRANSISTOR	26-12199- 3	21019	99	D	SM65	96214	TERM BRD ASS
TRANSISTOR	26-12223-801	20109	99	D	SM65	96214	
TRANSISTOR	2-00056-004	20712	83	F	SM65	03848	PLATF ALIGN
TRANSISTOR	2-03960-001	20405	83	F	SM65	03848	
TRANSISTOR	2N1048	20905	99	D	SM65	96214	
TRANSISTOR	2N109	20323	88	F	SM65	01294	DIFF AMPL
TRANSISTOR	2N109	20223	99	E	SM65	01294	
TRANSISTOR	2N1129	20109	9D	E	SM65	98329	MODULE
TRANSISTOR	2N1136	20103	9D	E	SM65	77068	MODULE
TRANSISTOR	2N1136	21217	9D	D	SM65	77068	DC XCIT
TRANSISTOR	2N1136	21217	9D	D	SM65	77068	DC XCIT
TRANSISTOR	2N1146	20221	99	E	SM65	98925	
TRANSISTOR	2N1150	20117	99	D	SM65	96214	
TRANSISTOR	2N1150	20123	99	D	SM65	96214	
TRANSISTOR	2N1152	20117	99	F	SM65	96214	
TRANSISTOR	2N1156	20108	99	D	SM65	96214	
TRANSISTOR	2N1232	20411	99	D	SM65		
TRANSISTOR	2N1232	20314	99	E	SM65		
TRANSISTOR	2N1232	20301	99	E	SM65		
TRANSISTOR	2N1232	20323	99	D	SM65		
TRANSISTOR	2N1232	20326	99	D	SM65		
TRANSISTOR	2N1232	20319	99	D	SM65		
TRANSISTOR	2N1232	20112	99	E	SM65	92400	
TRANSISTOR	2N1232	20124	99	E	SM65		
TRANSISTOR	2N1232	20124	99	E	SM65		
TRANSISTOR	2N1233	20305	99	F	SM65		
TRANSISTOR	2N1289	20108	99	D	SM65	24446	
TRANSISTOR	2N1301	21212	9D	E	SM65	01294	AMPLIFIER
TRANSISTOR	2N1301	21217	9D	E	SM65	01294	AMPL AUD FRE
TRANSISTOR	2N1302	20109	99	D	SM65	96214	
TRANSISTOR	2N1303	20109	99	E	SM65	96214	
TRANSISTOR	2N1310	20803	9D	F	SM65	04662	TV CTL UN
TRANSISTOR	2N1312	20214	9D	F	SM65	88431	DIST CAM UN
TRANSISTOR	2N1335	20927	99	D	SM65	01281	HF AMPL
TRANSISTOR	2N1358	20302	99	E	SM65	16764	
TRANSISTOR	2N1358	20302	99	E	SM65	16764	
TRANSISTOR	2N1374	20727	9D	E	SM65	01294	POWER SUPPLY
TRANSISTOR	2N1381	21031	9B	F	SM65	96214	PWR SUPPLY
TRANSISTOR	2N145	20104	99	D	SM65	96214	
TRANSISTOR	2N1482	20305	99	E	SM65	01294	
TRANSISTOR	2N1484	20315	99	E	SM65		
TRANSISTOR	2N158	20201	99	D	SM65	49956	
TRANSISTOR	2N1605	20212	9D	F	SM65	82219	MODULE
TRANSISTOR	2N1605	20215	9D	F	SM65		ECAN
TRANSISTOR	2N1605	20216	9D	F	SM65	82219	MODULE
TRANSISTOR	2N1605	20216	9D	F	SM65	82219	MODULE
TRANSISTOR	2N1605	20217	9D	F	SM65	82219	GEN

APPENDIX E

ERROR MESSAGES

The error conditions recognized by the program are indicated by the following messages:

FIRST SEARCH TAPE HAS NO ID FILE

A directory was not found at the beginning of the first reel of the search tape set. Execution is terminated.

INCORRECT CARD IS . . .

The user's input card that appears here is not a LIST, SORT, SEARCH, or END card, or has no * in column 1. Card is ignored.

YOU FORGOT THE SEARCH CARD

No search statement appears in the input deck.

DIRECTORY EXCEEDS TABLES

Directory file defines more report fields than can be processed by the program. Execution is terminated.

xxxxxx CARD . . UNKNOWN FIELD WILL BE IGNORED . . .

The field name which is printed to the right has been found on control card xxxxxx and is not a report field included in the search tape directory. Field is ignored.

YOU HAVE EXCEEDED THE MAXIMUM NUMBER OF ITEMS ON YOUR *xxxxxx CARD.

ONLY THE FIRST xx ITEMS WILL BE USED.

More than the allowable number of field names appear in a LIST or SORT card. (Limit is 2 for SORT card, 25 for LIST card.) Excess names are ignored.

NAME OR VALUE TOO LONG, READ xxx . . . .

The literals printed to the right appear within the search statement as either a field name or value containing too many characters. Execution is terminated.

**ERROR . . . SEARCH STATEMENT INCOMPLETE

The search statement has ended at an unexpected point, for example, F/I NAME = BULB AND. Execution is terminated.

**ERROR . . . UNKNOWN OPERATOR xxxxxx CARD READS . . .

xxxxxx appears in the context of a logical operator within the search statement, but is not AND, OR, or NOT. Execution is terminated.

**ERROR . . . MISSING DELIMITER, OPERATION READS . . .

One of the surrounding $ for an operator of the search statement is missing. If the operator is recognized, execution will continue.

**ERROR . . . CODE ITEM xx . . . IS UNKNOWN

The characters printed here were found as a field name within the search statement. The field name does not appear in the directory.

INCORRECT DATE IS xxx

A date has been found in the search statement which is not in the proper format, for example, FAIL DATE = 11/1764/. Execution is terminated.

**ERROR . . . CARD READING xxx . . . INCORRECT

The portion of the search statement which is printed here is meaningless. Execution is terminated.

SEARCH STATEMENT EXCEEDS TABLES

The search statement is too complex to be analyzed in the available table space.

**ERROR . . . PARENTHESES DO NOT BALANCE IN SEARCH

STATEMENT. NO PROCESSING.

Self-explanatory - execution is terminated.
There are five messages which may appear as part of the analysis of the search statement. They result from logic errors or redundancies within the search statements; they are the following:

ABOVE CONDITION IS IMPOSSIBLE

The aforementioned AND segment requires the truth of two mutually exclusive conditions. The segment is FALSE by definition.

ABOVE CONDITION IS ALWAYS TRUE

The aforementioned OR segment constitutes a logical tautology.

ABOVE CONDITION CAN BE SIMPLIFIED

This segment contains two or more requirements such that the truth of one implies the truth of another.

***SEARCH STATEMENT IS IMPOSSIBLE

The truth of the search statement requires an impossible condition.

***SEARCH STATEMENT IS ALWAYS TRUE

The search statement is a logical tautology.
When a segment is found to be always true or always false, the effect of that segment on the search statement as a whole is checked. This may reveal that the entire statement is either always true or always false. If this is so, execution is terminated. Otherwise, the search statement is optimized to prevent checking the segment whose value is known. and execution continues. Consider the following search statement:

FAIL DATE \$GT\$ 1/1/61 \$AND\$ ((F/I NAME = BOLT \$OR\$ F/I NAME

\$NE\$ BOLT) \$OR\$ (REPORT NO \$GT\$ 1100 \$OR\$ REPORT NO \$GT\$ 1200))

AND (F/I MFR = A AND F/I MFR NE A)

This would reduce to segments of the following form:

(001 FAIL DATE GT 1/1/61 AND (002 TRUE AND (005 TRUE

(002 (003 TRUE OR (004 TRUE

(003 F/I NAME = BOLT OR F/I NAME NE BOLT

(004 REPORT NO GT 1100 OR REPORT NO GT 1200

(005 F/I MFR = A AND F/I MFR NE A

When subroutine PARTL scans these segments beginning with segment 5, the following conditions will be detected:

(1) Segment 5 is impossible.
(2) Segment 4 can be simplified - it can be made simply REPORT NO GT 1100.
(3) Segment 3 is always true.
(4) Segment 2 must always be true since segment 3 is always true.
(5) Segment 1 is always false since segment 5 is always false.
(6) The search statement is recognized to be impossible and execution is terminated.

*LIST ITEMS EXCEED PAGE WIDTH. FIRST xxx WILL BE USED.

The report fields specified on the LIST card cannot be printed on a single output line. Only the first xxx will appear.

APPENDIX F

PROGRAM CONTROL SECTIONS

Subroutines

MAIN. - Subroutine MAIN performs the following:

(1) It reads and interprets the directory file of the search tapes.

(2) It reads user input cards and calls interpreting subroutine for each.

(3) It writes directory file on the hit tape followed by end of file.

(4) It calls subroutine SELECT to perform search.

(5) If the number of hits is less than 1000, it calls subroutine FORM to generate, from the user's LIST card, a format for tabular output.

(6) If sorted output has been requested, it calls subroutine SOUT to order and output results. Execution is terminated on return.

(7) If the output is not to be sorted, it calls subroutine TABL to print the contents of the hit tape. Execution is terminated on return.

WHICH1. - Subroutine WHICH1 performs the following:

(1) It is called by MAIN to interpret the contents of the SORT and LIST control cards.

(2) It extracts each field name from the control card (reading additional cards where necessary).

(3) It compares the extracted name to the list of recognized field names contained in array NOMEN (common block /STUFF/).

(4) If a field name is unknown, it is ignored and an error message is printed.

(5) If the field name is found in NOMEN, a pointer to the field will be stored in argument array INLIST and the card will be scanned for additional names.

(6) If the number of names on the card exceed the capacity of INLIST, excess names will be ignored and an error message printed.

(7) When the next control card (identified by * in column 1) is encountered, control returns to MAIN.

NORMAL. - Subroutine NORMAL performs the following:

(1) It reads the search cards and prints as read.

(2) It checks field names against those contained in the search tape directory.

(3) It reports errors in format of the card.

(4) It converts search statement into normal form of a logical algebraic expression. The normal form is constructed in array EQUATN (common block /LIST/).

(5) It calls subroutine LEVEL to continue analysis of the search statement. On return from LEVEL, control returns to MAIN.

LEVEL. - Subroutine LEVEL performs the following:

(1) It develops from the normal form of the search statement, a string of triples for array P (see Analysis of Search Statement).

(2) It removes unnecessary triples from P and orders remaining triples by C value. (Array P is stored in common block /PROD/.)

(3) It calls subroutine PARTL to continue analysis. On return from PARTL, control returns to NORMAL.

PARTL. - Subroutine PARTL performs the following:

(1) It scans array P once from bottom to top. During this pass, the following are performed:

(a) Each segment is printed out in expanded form using subroutine OUT. The segment numbers are converted to BCD form by subroutine BCD.

(b) Each triple is checked for logical consistency with other triples in its segment using subroutine LOGIC.

(c) Each segment is assigned a cumulative length. Triples are reordered within segments according to their operand. Triples with simpler operands are placed at the beginning of their segment.

(2) It constructs the STEP array (common block /LIST/).

(3) When STEP has been constructed, the analysis of the search statement is complete and control is returned to LEVEL.

BCD. - This subroutine converts a positive number less than 1000 to a six-character BCD string. The result is returned to the caller through labeled COMMON block /BCDWD/.

OUT. - Subroutine OUT is called by PARTL to print a line of the search statement analysis. The print line is prepared by PARTL in array PRINT, common block /PRINT/. On the first entry to OUT, the heading ANALYSIS OF SEARCH REQUEST precedes the print line. If the print line is the first line of a new segment -- as indicated by nonblank initial word -- a line will be skipped before printing.

LOGIC. - This subroutine is called by PARTL to check each triple whose operand is a term of the search statement (as opposed to the value of another segment). If another triple is found in the segment whose operand refers to the same report field as does this triple's, the two operands are checked for logical compatibility. If one of the operands is redundant, it is removed from the P array, the cumulative length of the segment is decremented, and a redundancy flag is set. If the two operands produce a logical tautology or inconsistency, the segment will be flagged as either true or false. When the triple has been checked against all other triples in its segment, control returns to PARTL (see appendix E for further comment on logic checking).

SELECT. - Subroutine SELECT performs the following:

(1) It is called by MAIN to control the search operation.

(2) It initiates search by calling subroutine READ (with argument = NOBUF) to fill the input buffer area (BUFFER) with report forms from the search tape.

(3) It checks each report form in turn against the search statement by the following methods:

(a) Checking conditions in the order specified by STEP (see Analysis of Search Statement)

(b) When a report field is required for checking, calling subroutine GET to extract its value from the packed report form; the extracted field is saved in case it should be referred to again by the search statement

(c) Checking the extracted value for the field against the value specified in the search statement; if only part of the value is to be checked, subroutine QUALFY is called

(4) It calls subroutine ADD to copy each report form that matches the search statement.

(5) It calls subroutine READ (with argument = 1) to read more report forms from the search tape as soon as all forms from a section of BUFFER have been checked.

(6) It calls subroutine ADD (with argument = 0) when the entire search tape has been checked. Control is then returned to MAIN.

READ. - Subroutine READ performs the following:

(1) It is called by SELECT to read report forms from the search tape.

(2) When argument is 1, reports will be read into the section of buffer just checked by SELECT.

(3) When argument is not 1, the entire buffer area will be filled. This is assumed to be the first entry to READ; therefore, READ also checks the first record read to be sure it contains the number of report forms of the size expected.

(4) It records the number of words read into each buffer section.

(5) If an end of file is encountered, it is skipped and reading continues from the next file.

(6) If an end of tape is encountered, reading continues from the next tape.

(7) If an end of set is encountered, the word count for the first empty buffer section is set to -1 and the pointer to the next section is set to 0.

(8) Control returns to SELECT.

GET. - This subroutine is called by SELECT to extract a particular field (field LEFT) from a report form. The extracted field is placed in array BUF (common block /LIST/), and its beginning location in BUF is placed in word LEFT of array HOLD (common block /PULL/). Word LEFT of array HAVGOT (common block /PULL/) is set equal to the sequence number of the report form. This enables SELECT to recognize that the field has already been extracted and its value stored if the field should be referred to again by the search statement. When the field is extracted, blanks are removed and the value is right-adjusted.

QUALFY. - Subroutine QUALFY is called by SELECT to check a report field against a partial value. It masks off the section of the field to be compared and checks against the N characters of the search value. (If the value of the field from the report form does not contain N nonblank characters, it will be declared to be less than the search value.)

ADD. - Subroutine ADD is called by SELECT. If argument $\neq 0$, a report form which matches the search statement has been found. The hit count NOHIT (common block /RESULT/) is incremented, and the report form is copied into array OUTBUF. As soon as OUTBUF is full, it is written onto the hit tape.

If the argument = 0, the search is complete. Subroutine ADD prints out the total number of hits found, writes the end of set message on the hit tape, rewinds the hit tape, and returns control to SELECT.

FORM. - Subroutine FORM is called by MAIN to supervise the generation of FORTRAN FORMAT statements to be used in printing the search output.

FORM calls the generator FORMAT once for each field on the LIST card. IF FORMAT detects that the requested fields would exceed the page width, an error message is printed, and the excess fields will not appear in the output. Two formats are produced, TFMT which is used for the heading line, and FMT which is used to print the report fields. When all fields have been considered, FORM inserts the closing right parentheses and returns control to MAIN.

FORMAT. - Subroutine FORMAT is called by subroutine FORM to generate FORMAT specifications for printing the name and value of a report field. The specifications are added to the current contents of array TFMT and FMT, respectively. In constructing the specification, FORMAT does the following:

(1) Includes the appropriate number of skipped columns in either the title or value specification in order to keep the two formats in alinement.

(2) Attempts to break the field name into as many as three parts when the name width exceeds the value width by eight characters or more. The parts will appear in the heading one beneath another.

(3) Maintains a count of the number of print positions used (in variable KOLUMN). If it is determined that a new field will cause KOLUMN to exceed MAXWID (set in FORMAT), the logical variable NOMOR is set to TRUE and control returned to FORM. (For the application described in this report, an exception is made for overflow caused by the three description fields, DESC LINE1, DESC LINE2, and DESC LINE3. If they are requested as the final fields on the LIST card, they will be printed as a second and third line of report output, with no title provided.

Subroutine BCD is used to convert numeric parts of the FORMAT to BCD form. The two arrays, TFMT and FMT, are initially set to blanks except for the first word of each, which is initialized to contain a carriage control character.

SOUT. - Subroutine SOUT performs the following:

(1) It is called by MAIN to control the ordering and output of the tabular reports requested on the LIST and SORT control cards.

(2) SOUT defines the following arrays for the sort field(s) and each of the fields to be listed:

(a) WORD - ending word location within the report form of sort fields or the beginning word location within the report form of fields to be listed

(b) SHIFT - number of bit positions the ending word must be shifted right to right-adjust the field (for sort fields), or (for listing fields) the number of bit positions the beginning word must be shifted left to left-adjust the field

(c) LONG - the length of the field in bits (for sort fields) or in words (for listing fields).

(3) It records (in LENGTH) the number of words required to hold all extracted and adjusted sort and listing fields from a report form.

(4) It determines (in WORD2) the number of records from the hit tape which can be sorted in the available storage. This is based upon the size of BUFFER (set at 15 000 words), LENGTH, and LRLREC (the number of report forms per hit tape record).

(5) It determines, if sufficient storage is not available to sort all the hits at once, the number of batches of sorted output which will be required. Since the program is not designed to generate large amounts of output, no provision is made for merging the sorted batches. If the number of hits exceeds the number of report forms which can be sorted internally, the output will appear in several batches - each of which is ordered, but has no order relation to other batches. Note that the number of reports which can be sorted internally and, consequently, the number of batches into which the output will be divided, is a function of the number and size of report fields on the LIST and SORT cards.

(6) It reads, from the hit tape, the WORD2 records to be sorted. As each record is read, the required fields are extracted from all its report forms using the information in arrays WORD, SHIFT, and LONG. Those fields which did not appear on the LIST or SORT cards are discarded.

(7) It defines arrays LIST1 and LIST2 in the following manner: As the sort field of each report form is extracted, it is compared to the sort field of the previous report form. In this way, consecutive report forms whose sort fields happen to be ordered as they appear on the hit tape are identified. The location, within BUFFER, of the first member of the Ith such group of report forms is recorded in word 1 of LIST1 and word I-1 of LIST2 (except that no entry is made in LIST2 for I = 1).

When the WORD2 records have been read and their contents prepared as described previously, SOUT calls subroutine SORTER to perform ordering and output. When the process is complete for all batches, control is returned to MAIN.

SORTER. - Subroutine SORTER is called by subroutine SOUT to supervise the ordered output of the search results. The order of output is determined as described in the section Output Ordering. SORTER calls subroutine OUTPUT to print the listing fields of each report form in sequence. When output is complete, control is returned to SOUT.

OUTPUT. - Subroutine OUTPUT is called to print the requested fields from a report form. The two format arrays, TFMT and FMT, which were generated by FORMAT are used. OUTPUT maintains a count of the number of lines printed per page (variable LINEKT). When the maximum number permitted per page (variable NOLINE set in FORM) has been printed, a new page is started with the heading lines repeated.

TABL. - Subroutine TABL is called by MAIN to supervise the printing of unsorted output. It first builds the three arrays, LONG, SHIFT, and WORD, described under subroutine FORM. It then begins reading records from the hit tape. When enough records have been read to fill the BUFFER area, the accumulated report forms are processed as follows:

The fields requested on the LIST card are extracted and adjusted using arrays LONG, SHIFT, and WORD. The extracted fields from a report form are stored in HOLD, and subroutine OUTPUT is called to print them from there.

When all hit forms have been printed, control returns to MAIN.

Principal Named Common Blocks

The principal named common blocks are the following:

/MESG/ (3 words (defined in MAIN)):

EOF	end of file message
EOS	end of set message
EOT	end of tape message (see definition in appendix B)

/OUTTAP/ (1 word):

OTAPE	tape number to be used as the hit tape (defined in MAIN)

/RECORD/ (5 words):

LNGABS	length (in words) of each report form on search tape
LNREC	number of report forms per record on the search tapes
LRABS	length (in words) of each report form as it will be copied on the hit tape

LRLREC	number of report forms per record as they will be packed on the hit tape
RECKNT	pointer to front of report form - modified during execution

/FILES/ (7 words):

FILE	pointer to tape currently being searched
SEARCH(5)	the tape numbers assigned to the search tapes (defined in MAIN)
TAPE	tape number currently being searched

/NOTAPE/ (1 word):

MAXTAP	(set in MAIN) defines maximum number of reels for the search file

/VALUS/ contains the NOUN, PRED, OBJ, and TABL arrays as described in the section Analysis of Search Statement

/FIELD/ contains the LNGV and LNGN arrays which give the number of bits in a report field and the length, in bits, of the field name, respectively

/LIST/ (1422 words):

In subroutines NORMAL, LEVEL, and LOGIC, the words are used as follows:

I	a pointer to the last entry in array EQUATN
EQUATN	750-word array contains the normal form of the search statement
LIST	320-word array used as scratch area during analysis
KVEC	350-word array used as scratch area during analysis
JMAX	scratch variable used during analysis

In subroutines PARTL, TABLE, SELECT, and GET, the words are allocated as follows:

I	used as scratch variable
STEP	400-word array described in section Analysis of Search Statement
HOLD	50-word array that points to an extracted field's location in BUF

BUF	450-word BUF used as a temporary storage area for extracted fields during report checking
IHOLD	pointer to next available location in BUF
words 903 to 1422	scratch area

/BUFFER/ (5708 words):

LREC(6)	variables give the number of data words in each section of BUFFER
BUFFER(950, 6)	used as input BUFFER area for report forms
IBUF	points to the section of BUFFER which contains the report form currently being checked
NOBUF	constant (set at 6 in SELECT) which defines the number of sections in BUFFER

/TAPES/ (2 words (set in MAIN)):

IOUT	number of the system output unit (set to 6)
INTAPE	number of the system input unit (set to 5)

/FORMS/ (204 words):

LAST	(set in FORMAT) specifies the major dimension of arrays FMT and TFMT following
I1	number of words used in column 1 of TFMT
I2	number of words used in column 2 of TFMT
I3	number of words used in column 3 of TFMT
TFMT	150-word array which contains the FORMAT statement for printing the search output heading; its three columns permit the heading to be three lines long
FMT	50-word array which contains the FORMAT statement for printing the report forms

/SORTC/ (18 025 words):

K	number of naturally ordered groups of report forms to be sorted
INTCNT	number of report forms (in a batch) to be sorted
KLEAR	number of words in the sort field(s) of each report form

LIST1(1001)	array defined in description of SOUT
LIST2(1001)	array defined in description of SOUT
BUFFER(15 000)	storage area for sort and listing fields of report forms to be sorted
ISAVE(1001)	see section Output Ordering
LENGTH	number of words required for sort and listing fields from each report form
PREV(16)	scratch area
BEGIN	scratch variable
MANY	number of words in listing fields

REFERENCE

1. Sheridan, Peter B.: The Arithmetic Translator-Compiler of the IBM Fortran Automatic Coding System. Comm. Assoc. Comp. Mach., vol. 2, no. 2, Feb. 1959, pp. 9-21.

NASA-Langley, 1967 —— 9 E-3383

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

—NATIONAL AERONAUTICS AND SPACE ACT OF 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS: Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES: Information less broad in scope but nevertheless of importance as a contribution to existing knowledge.

TECHNICAL MEMORANDUMS: Information receiving limited distribution because of preliminary data, security classification, or other reasons.

CONTRACTOR REPORTS: Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS: Information derived from or of value to NASA activities. Publications include conference proceedings, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Notes, and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION DIVISION

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Washington, D.C. 20546